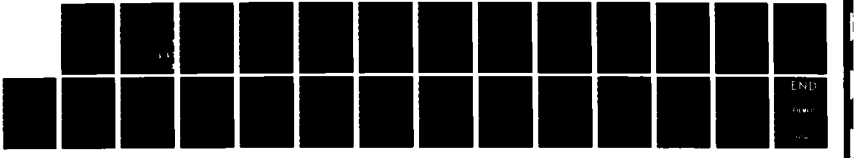
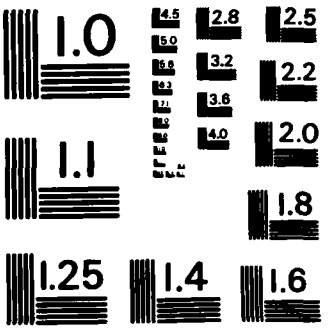
AD-A160 148 PARALLEL DIRECT METHODS FOR SOLVING BANDED LINEAR SYSTEMS(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE Y SAAD ET AL. AUG 85 YALEU/DCS/RR-307 1/1

UNCLASSIFIED N00014-82-K-0184 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

AD-A160 148

Parallel direct methods for solving banded linear systems

Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-387
August 1985

DTIC FILE COPY

This document has been approved for public release and sale; its distribution is unlimited.

DTIC ELECTE OCT 2 1985 S D A

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

85 09 17 139

Abstract. In this paper we propose several implementations of Gaussian elimination for solving banded linear systems on multiprocessors. Three simple architectures are considered: a multiprocessor ring, a grid array and a hypercube. Our complexity analysis fully accounts for communication delays by using simple models where both latency and actual transfer times are incorporated. When the number of processors is small relative to the bandwidth of the system a row interleaved implementation of Gaussian elimination algorithm is attractive. Otherwise, a two–dimensional grid is essential for achieving higher speed-up. The hypercube architecture gives the smallest communication latency times.

Parallel direct methods for solving banded linear systems

Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-387
August 1985

DTIC
ELECTE
OCT 2 1985
A

This work was supported in part by ONR grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston

1. Introduction

Symmetric, positive definite, banded linear systems constitute one of the most important classes of linear systems encountered in scientific computing. In this paper we propose and analyze several implementations of Gauss elimination for dealing with such systems on parallel computers. The implementations are developed and compared for three simple multiprocessor architectures, namely a ring of processors, cf., Figure 1, a rectangular two-dimensional grid of processors, cf., Figure 2, and a hypercube of processors, cf., [13].

The banded linear systems treated in scientific computing are often very large with large bandwidths. For problems arising from the discretization of elliptic partial differential equations in two variables, the bandwidth of the matrix is usually of the order of $\sqrt{N}$ if the system is of order N. For this reason we will be interested in cases where the number of processors is smaller than the bandwidth of A, and in the less restrictive cases where it is smaller than the square of the bandwidth. In either of these two cases there is inherent parallelism in the Gaussian elimination algorithm that can be exploited and our focus will be on developing effective implementations of this simple algorithm. We will not consider the case where the number of processors is very large compared to the bandwidth, such as for example when the matrix is tridiagonal. This case can be treated by special techniques such as cyclic reduction [4] or a substructured banded elimininations [2, 11]. See [5] for a general discussion of this case.

The main assumptions on the architecture are that each processor has its own memory and holds its share of the data. Data is moved from one processor by message passing using the local links of the array. No data transfer is made via shared global memory, or via a bus.

In looking for effective parallel algorithms we are guided by both arithmetic efficiency and communication efficiency. The timing models used for analyzing complexity assume that communication time is nonnegligible as compared with arithmetic. Moreover, for generality, a start-up time is incorporated whenever estimating the time for performing either a data communication task or an arithmetic task. We assume for the sake of simplicity of our analysis that communication and arithmetic are not overlapped.

As will be seen, an important limiting factor in nearest neighbor arrays is the start-up in communication. Intuitively, when the number of processors increases, the packets of data which are spread to a larger number of processors must travel distances that become larger and larger while the packets become smaller. The result is an inevitable increase in data movement start-ups and hence a serious obstacle to speed-up for very large number of processors. We are then lead to the following paradoxical question: given an arbitrarily large number of processors is it best to use all of the available processors or to "turn-off" a few of them and use only part of the available computing power? Clearly, the question relates to one fixed algorithm otherwise an easy answer would be to switch to a different method (if one exists) that will take better advantage of the large number of processors.

We will see, for example, that for a class of implementations of Gaussian elimination on multiprocessor rings or grids the best computing time is not realized for the maximum allowable number of processors when N is large. Thus, the best time that can be achieved on a grid of processors is of the order of $O(\nu^{2/3}N)$ where ν is the half-bandwidth of the matrix. These algorithms can be termed lock-step algorithms, as they consist of executing the outer loops of Gauss' algorithm in succession without overlapping them. A different approach, the wavefront implementation, will also be described and compared with the simpler lock-step algorithms. Our comparison shows that while the wavefront approach may be superior when the number of processors is large, this is not true for small number of processors.

We will compare the performance of the Gaussian elimination algorithms on the three different architectures. In the case of a small number of processors, we may elect to map the ring into the

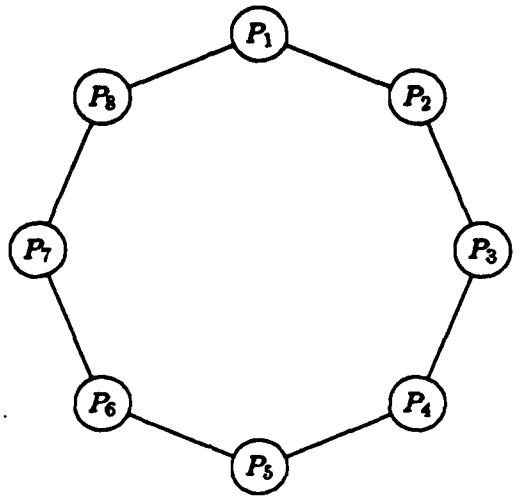


Figure 1: An 8-processor ring.

grid or hypercube networks and use an algorithm that is efficient on the ring. In this case, it is not a priori clear that the algorithms designed specifically for the grid or hypercube will outperform the algorithm mapped from the ring. However, we will see that the performances are comparable when the number of processors is small.

2. The three models and their properties

2.1. Description of the models

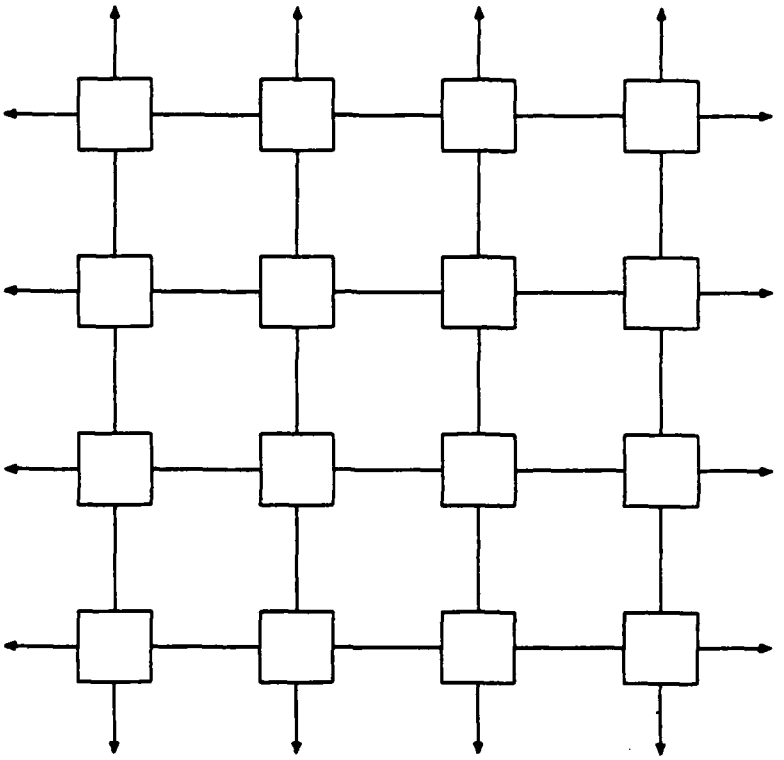
Consider the linear system $Ax = f$, where A is a real N x N matrix whose half-bandwidth is ν, i.e., we have $a_{i,j} = 0, \forall i,j : |i-j| \geq \nu$, and whose total bandwidth is $2\nu - 1$. We will make the important assumption that A is such that no pivoting is necessary in Gaussian elimination. We would like to solve the above linear system on a multiprocessor consisting an ensemble of k identical processors, each with its own memory sufficiently large to hold its equal share of the data. The processors are interconnected according to one of the following three simple schemes:

1. The first architecture consists of a nearest neighbor interconnection ring, see Figure 1. The processors are numbered consecutively from 1 to k. We assume that a vector of length m can be sent from one processor to one of its neighbors via the local link in time equal to

$$\beta_R + m\tau_R, \tag{2.1}$$

where β_R is a constant latency time, and τ_R is the elemental transfer time of the local links, in seconds per word. We assume that any processor can send (or receive) a data item from one neighbor while sending (or receiving) another data item from the other neighbor.

2. The second architecture consists of a two-dimensional grid of k processors arranged on a square grid with $\sqrt{k}$ processors on each side, see Figure 2. For convenience we assume that the processors on each side of the grid are connected to those on the opposite side. We assume these wrap around connections in order to yield more homogeneous complexity results. These connections are not essential for the algorithms themselves. We will often use the term $2-D$ array for this scheme. Our assumptions are identical with those of the ring model but one processor is now able to simultaneously communicate with four neighbors. We assume that



Acces... For
NTIS CRA
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1
OTIC COPY INSPECTED

Figure 2: A 4 x 4 multiprocessor grid.

the time required to send a vector of length m from one processor to one of its neighbors is modeled by

$$\beta_G + m\tau_G, \tag{2.2}$$

where β_G is the grid latency time, and τ_G is the grid elemental transfer time.

3. Finally, the third architecture consists of a hypercube, or n–cube $k = 2^n$ identical processors, numbered from 0 to $2^n - 1$. The interconnection in the hypercube is such that there is a link between two processors if and only if the binary representations of their numbers differ by one and only one bit. Thus any node has exactly n neighbors. For example, the 8 nodes of the 3-dimensional cube ($n = 3$) can be represented as the vertices of a three dimensional hypercube, as is shown in Figure 3. We assume that, it takes the time

$$\beta_H + m\tau_H, \tag{2.3}$$

to transfer a vector of length m from one processor to any number of its n neighbors. Moreover, we assume as before that communication in all of the n directions can take place simultaneously.

In Ipsen, Saad and Schultz [3], the formula (2.1) was used for estimating communication times in various Gaussian elimination implementations for solving dense linear systems. We remark that latency times may be much larger than elemental transfer times.

For arithmetic computations, we assume that for all architectures a sequence of m pairs of arithmetic operations involving an addition and a multiplication (such as the inner product of 2 m-vectors) takes the time

$$\gamma + m\omega, \tag{2.4}$$

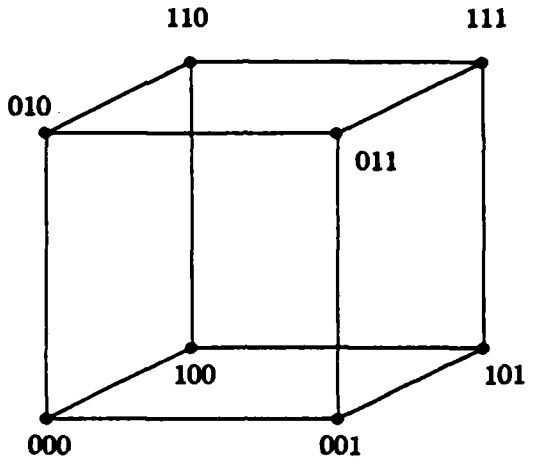


Figure 3: 3-D view of the 3-cube.

in any of the k processors. Formula (2.4) allows for the case in which each node of the multiprocessor can itself be a pipelined or vector machine, in which a sequence of operations is done more efficiently if the number of operations is large. In case each processor is not a pipelined machine we have $\gamma = 0$.

In order to simplify our analysis we will assume throughout the paper that one cannot overlap arithmetic with communication in any processor. The reason for the non-overlapping assumption is essentially pedagogical and can be justified as follows. Models where arithmetic and communication can be overlapped, are more complicated to analyze. Yet observe that the execution time of an algorithm in an environment where overlapping is assumed is within a factor of ***only two*** of that same algorithm run in an environment where no overlapping is assumed. Indeed, consider any algorithm and let t_A be the total time required to perform its arithmetic and t_T be the total time required to perform its transfers. In the overlapping case, the total execution time t_E will be *at least* $max\{t_A, t_T\}$ which represents the time with maximum overlapping. In the non-overlapping case, it will be simply $t_A + t_T$ which satisfies

$$\frac{1}{2}(t_A + t_T) \leq max\{t_A, t_T\} \leq t_E \leq t_A + t_T.$$

2.2. Data Transfer Operations in Gaussian Elimination

A particular data transfer operation which is essential in Gaussian elimination is that of sending a vector of m elements from one processor to all the others or to a few neighboring processors. As was pointed out in [12, 3], an efficient way of achieving this type of data movement, is to pipeline the data as follows. Assume that the data is to be moved from Processor P_1 to the sequence of i

consecutive processors $P_2, P_3..P_{i+1}$, i.e., through a path of length i and let us split the data into μ packets of equal size. Then in step 1 the first packet is sent from P_1 to P_2. In step 2, while sending the first packet to P_3, P_2 receives the second packet from P_1. Generally, at step j, the first packet reaches P_{j+1} from P_j while the second packet follows from P_{j-1} to P_j, etc. The first packet reaches P_{i+1} at the i^{th} step and $\mu - 1$ more steps are needed for the remaining packets to reach P_{i+1}, resulting in a total of $(\mu - 1) + i$ steps and a time of

$$t(\mu) = (\mu - 1 + i)(\beta + \frac{m}{\mu}\tau) \tag{2.5}$$

where β, τ stand for β_R, τ_R (ring) β_G, τ_G (2–D array grid), or β_H, τ_H (hypercube). We remark that a ring can always be embedded in the 2–D array grid and hypercube. The above time is minimized for the optimal number of packets equal to

$$\mu_{opt}(m, i) = \sqrt{(i-1)m\frac{\tau}{\beta}}, \tag{2.6}$$

and the corresponding optimal time is

$$t_{opt}(m, i) = \left(\sqrt{m\tau} + \sqrt{(i-1)\beta}\right)^2. \tag{2.7}$$

Note that since we must have $1 \leq \mu \leq m$, formula (2.6) is valid only when

$$\frac{1}{m(i-1)} \leq \frac{\tau}{\beta} \leq \frac{m}{i-1}.$$

This will always be the case if m is large enough with respect with the ratio τ/β, which is usually small. However, in case $1/[m(i-1)] > \tau/\beta$ then the optimal number of packets is $\mu = 1$ and the above timing formula becomes $t_{opt}(m, i) = i(m\tau + \beta)$. When $\tau/\beta > m/(i-1)$ then the optimal number of packets is $\mu = m$ and the optimal time becomes $t_{opt}(m, i) = (m + i - 1)(\tau + \beta)$.

Formula (2.7) expresses the time in which a vector of size m can be moved ***along a chosen path of length*** i between two processors. This may not be the best possible time to transfer data between two processors because several parallel paths (i.e., paths that do not cross each other) can be used simultaneously to perform the data transfer. For the ring there are two parallel paths between any two processors, while for the 2-D grid there may exist up to four such paths. For the hypercube it can be shown that n parallel paths exist between any two given nodes [8].

Suppose that we want to transfer a data block of size m from some processor to all the processors in a ring. An efficient way of doing this operation is to pipeline the data in both directions around the ring from the initial processor. Then to reach every processor, the data must travel across only $i = \lceil (k-1)/2 \rceil$ processors in each of the two directions of travel. Therefore, broadcasting a vector of length m in a ring requires the time:

$$t_{Broadc}(m, R) \approx \left(\sqrt{m\tau_R} + \sqrt{\frac{k}{2}\beta_R}\right)^2. \tag{2.8}$$

Consider now the same operation of transferring a vector of length m from one processor to all others in an n-cube. At first, let us describe a simple algorithm for sending *one* element from one processor to all the other processors of the cube. The algorithm is based on the fact that any n–cube is the union of two $(n-1)$-cubes whose nodes are connected in a one-to-one fashion. In fact

this constitutes a simple recursive definition of n-cubes. Specifically, all the nodes whose binary number is of the form $0a$ where a is any $(n-1)$-bit binary number representing the vertices of an $(n-1)$-cube are linked to the nodes whose numbers are $1a$. This suggest the following algorithm for sending one element from node 0 to all other nodes in an n-cube. The powers refer to concatenation.

ALGORITHM: Hypercube Broadcast

(1) *Start:* Send data item from processor 0^n to processor $0^{n-1}1$.

(2) *Loop:* For $j = 2,3,\ldots n$ do

All processors numbered $0^{n-j+1}a_j$, where a_j is any $(j-1)$-bit binary number move data item in parallel to processor $0^{n-j}1a_j$.

Since there are n steps in the above algorithm, any given element can be transferred to all others in an n–cube in a time $n(\beta_H + \tau_H)$. When sending a vector of length m it is clear that the above idea of pipelining the data transfer can still be used. Since this will amount to pipelining the data transfer in a path of length n, it is clear that formula (2.7) applies, i.e., the optimal time for sending a vector of length m from one processor to all others in an n–cube is given by:

$$t_{Broadc}(m,H) \approx \left(\sqrt{m\tau_H} + \sqrt{(n-1)\beta_H}\right)^2. \tag{2.9}$$

3. Algorithms for the ring architecture.

The algorithms we develop in this section are valid only for the case when the number of processors is smaller than the bandwidth of the problem ($k \leq \nu$). We assume a ring interconnection. However, it is clear that since a ring can be mapped into many other architectures these algorithms have a fairly wide range of application whenever $k \leq \nu$.

3.1. Interleaved Gaussian Elimination

In [3] several modified Gaussian elimination algorithms for solving dense linear systems have been proposed. One of the more efficient algorithms, named Row-Scattered Gaussian elimination, consists in interleaving the system across the processors and performing a slight variation of the usual Gaussian algorithm.

In order to adapt this interleaved Gaussian elimination algorithm to banded linear systems, we distribute the equations of $Ax = f$ among the processors in the way shown in Figure 4. The i^{th} processor contains the N/k equations $i, i+k, i+2k, ..., i+N-k$, where we assume for convenience that N is a multiple of k.

Typically, at the i^{th} step of Gaussian elimination, the pivoting row (of length ν) is first sent to all processors and then each processor eliminates those rows among the rows $i+1, i+2, ..i+\nu-1$ that it holds. There are at most $\lceil \frac{\nu}{k} \rceil$ such rows in each processor. In order to keep the processors busy as much as possible and to obtain k x k diagonal blocks that have the nice property of being diagonal matrices, we employ the modification of the classical Gaussian elimination suggested in [3], i.e., eliminate the i–th variable not only rows $i+1, i+2, ..i+\nu-1$ but also in rows $i-1, i-2, .., q$, where $q = \lfloor \frac{N}{k} \rfloor$. This will result in a matrix that is upper triangular, banded, and with diagonal k x k

diagonal blocks (referred to as DDB matrices). The resulting algorithm is referred to as Algorithm RIBGE (Row-Interleaved Banded Gaussian Elimination) and is formally described below.

ALGORITHM RIBGE (Row–Interleaved Banded Gaussian Elimination)

For $q = 1, 2, \ldots N/k$ do:
For $i = 1, 2, \ldots k$ do:

(1) *Send pivot row:* Send row number $(q-1)k+i$ from processor i to all other processors.

(2) *Perform eliminations:*

In each processor $P_j, j = 1, 2, ..k$ do in parallel:
For $h = (q-1)k+j$, Step k, while $\{h \leq i+\nu$ and $j \neq i\}$ Do:

$$a_{h,*} := a_{h,*} - \gamma_{h,j} a_{qk+i,*},$$

where $\gamma_{h,j} = a_{h,(q-1)k+j} / a_{(q-1)k+i,(q-1)k+i}$.

Let us estimate the time to perform the Gaussian elimination process. For each step $i = 1, 2, ..N-1$, we need

- to transfer a row of size ν to all processors at a cost of (2.8):

$$\left(\sqrt{\nu\tau_R} + \sqrt{\frac{k}{2}\beta_R}\right)^2$$

- perform $\lceil \frac{\nu-1}{k} \rceil$ eliminations at the total cost of $\lceil \frac{\nu-1}{k} \rceil [\nu\omega + \gamma]$.

Summing up the above times for $i = 1, 2, ..N-1$ results in the total of approximately

$$t_{T,R} \approx N \left(\sqrt{\nu\tau_R} + \sqrt{\frac{k}{2}\beta_R}\right)^2 \tag{3.1}$$

for communication and

$$t_A \approx N \lceil \frac{\nu-1}{k} \rceil (\nu\omega + \gamma) \tag{3.2}$$

for arithmetic. These results will now be recast in a form that will be convenient for subsequent comparisons.

Proposition 3.1. *The total time required for performing the Row-Interleaved Banded Gaussian elimination on a k-processor ring is approximately:*

$$t_{RIBGE} \approx N \lceil \frac{\nu-1}{k} \rceil (\nu\omega + \gamma) + N\nu\tau_R (1 + \kappa\alpha_R)^2 \tag{3.3}$$

$$= N \left[\lceil \frac{\nu-1}{k} \rceil (\nu\omega + \gamma) \right] + N \left[\nu\tau_R + \sqrt{2k\nu\beta_R\tau_R} + \frac{1}{2}k\beta_R \right],$$

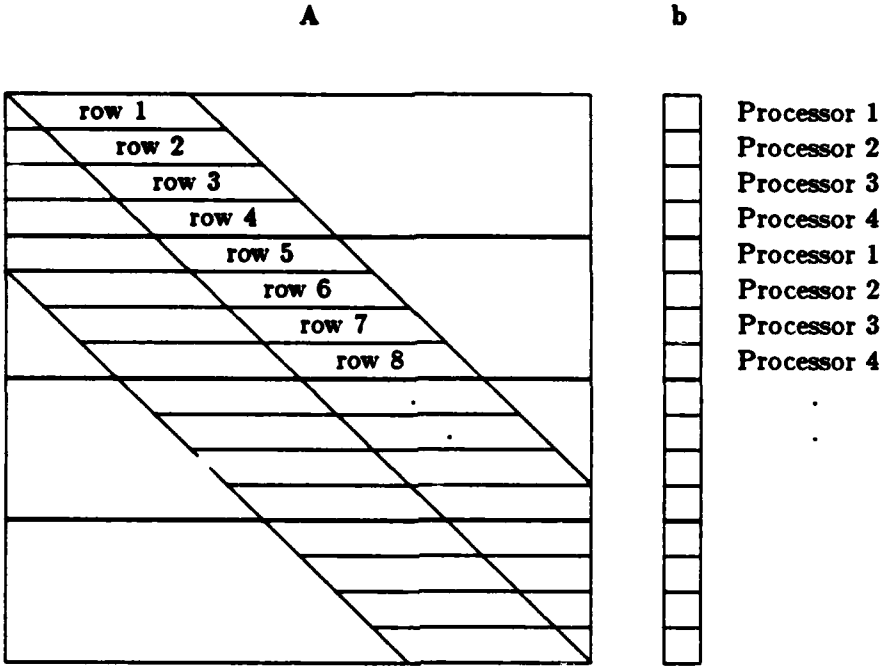


Figure 4: Row–wise interleaving of a banded linear system in a 4 processor system.

where $\kappa \equiv \sqrt{k}$ *and* $\alpha_R = \sqrt{\frac{\beta_R}{2\nu\tau_R}}$.

The approximation $(k-1)/k \approx 1$ has been made to derive (3.1) and (3.2). The above estimates indicate that for large enough bandwidth the time for arithmetic is reduced by a factor of k, as compared with the time on a single processor. When $\nu \leq k$, then we ***cannot*** further speed-up the arithmetic with the above algorithm, as is indicated by formula (3.3). Clearly, we will then be doing better with $k = \nu$ processors because of the increase of communication time.

Thus using an ever larger number of processors will not always speed-up the algorithm i.e., there is an upper limit beyond which it is ineffective to use more processors. This suggests that there is an optimal number of processors which can be obtained by trying to minimize the timing formula (3.3) with respect to the parameter k. Unfortunately, this function of k is complicated and its exact minimum even if available might be difficult to exploit and interpret. In order to simplify the expression we first replace the term $\lceil(\nu-1)/k\rceil$ by the approximation ν/k. We call $\tilde{t}_{RIBGE}$ the resulting right hand side of (3.3). A second simplification is realized by using an upper and lower bound for the term corresponding to communication time, namely

$$\nu\tau_R + \frac{k}{2}\beta_R \leq \left(\sqrt{\nu\tau_R} + \sqrt{\frac{k}{2}\beta_R}\right)^2 \leq 2\nu\tau_R + k\beta_R. \tag{3.4}$$

Note that a similar expression has been used in [3]. The upper bound simply corresponds to non-optimal pipeling of data transfers, whereby one splits the data into $\frac{k}{2}$ packets instead of the optimal μ_{opt} packets as given by (2.6).

Thus the time $\tilde{t}_{RIBGE}$ is bounded above by

$$\tilde{t}_{RIBGE} \leq \bar{t}(k) \equiv N\left[\frac{\nu}{k}(\nu\omega+\gamma)+2(\nu\tau_R+\frac{k}{2}\beta_R)\right]. \tag{3.5}$$

By differentiating the above function with respect to k and equating the result to zero, we find that the minimum of the above upper-bound is achieved at

$$k^* = \nu\sqrt{\frac{\omega+\gamma/\nu}{\beta_R}}. \tag{3.6}$$

Substituting k^* in (3.5) we get the optimal time

$$\bar{t}(k^*) = 2N\nu\left[\sqrt{(\omega+\gamma/\nu)\beta_R}+\tau_R\right]. \tag{3.7}$$

Denoting by k_{opt} the value of k for which the minimum of $\tilde{t}_{RIBGE}$ is achieved, we have

$$\min_k \tilde{t}_{RIBGE} = \tilde{t}_{RIBGE}(k_{opt}) \leq \tilde{t}_{RIBGE}(k^*) \leq \bar{t}(k^*). \tag{3.8}$$

A similar argument can be used for the lower bound

$$\tilde{t}_{RIBGE} \geq \underline{t}(k) \equiv N\left[\frac{\nu}{k}(\nu\omega+\gamma)+(\nu\tau_R+\frac{k}{2}\beta_R)\right], \tag{3.9}$$

the minimum of which is achieved for

$$k_* = \nu\sqrt{\frac{2(\omega+\gamma/\nu)}{\beta_R}} \tag{3.10}$$

and has the value

$$\underline{t}(k_*) = 2N\nu\left[\sqrt{\frac{1}{2}(\omega+\gamma/\nu)\beta_R}+\frac{1}{2}\tau_R\right]. \tag{3.11}$$

Moreover,

$$\min_k \tilde{t}_{RIBGE} = \tilde{t}_{RIBGE}(k_{opt}) \geq \underline{t}(k_{opt}) \geq \underline{t}(k_*). \tag{3.12}$$

Grouping the two bounds (3.8) and (3.12) we get the following result.

Proposition 3.2. *The approximate minimum time in which Algorithm RIBGE can be performed on a multiprocessor ring with an arbitrary number of processors is such that*

$$2N\nu\left[\sqrt{\frac{1}{2}(\omega+\gamma/\nu)\beta_R}+\frac{1}{2}\tau_R\right] \leq \min_k \tilde{t}_{RIBGE} \leq 2N\nu\left[\sqrt{(\omega+\gamma/\nu)\beta_R}+\tau_R\right]. \tag{3.13}$$

This result indicates that, for large values of ν, the optimal time is nearly linear in ν.

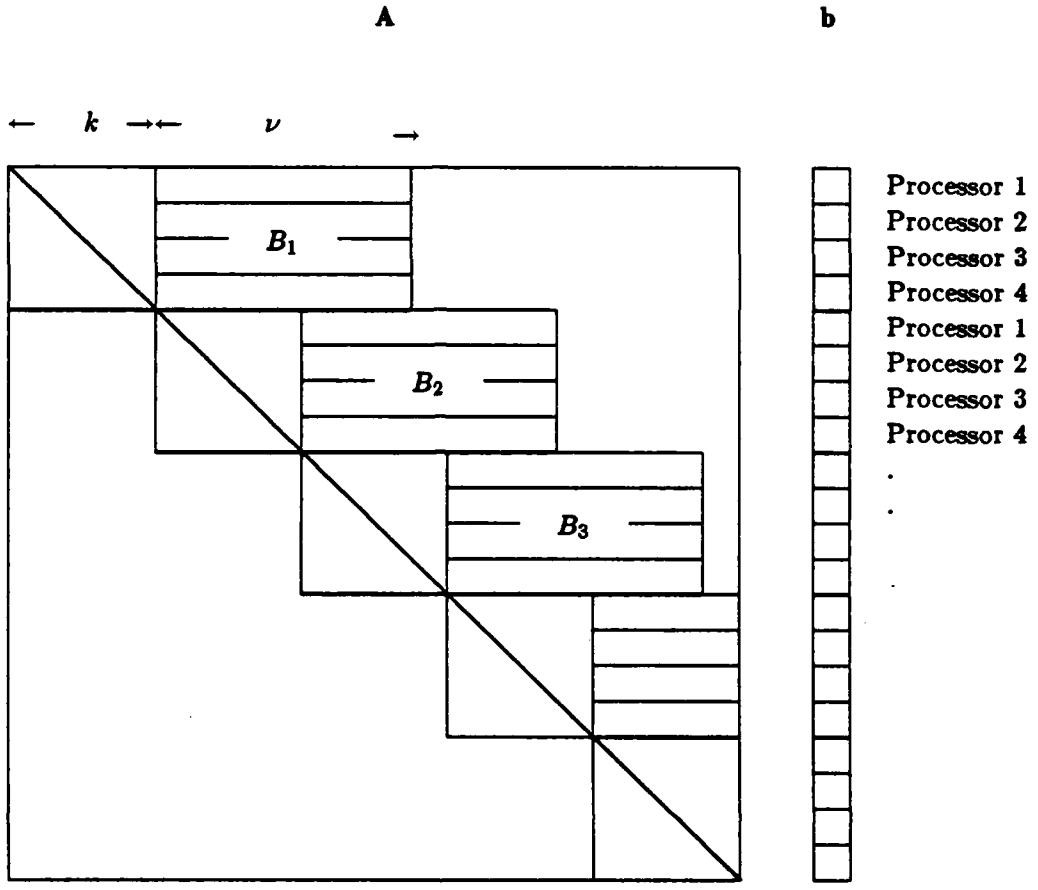


Figure 5: DDB banded upper triangular system resulting from interleaved Gaussian elimination in a 4-processor system.

3.2. Solution of the Resulting Triangular Systems

The upper triangular systems resulting from the Gaussian elimination algorithm described in Section 3.1 have the form depicted in Figure 5. Each diagonal block is a k x k diagonal matrix, while the off-diagonal blocks B_j are dense. The solution algorithm for such a system can be adapted from [3] in a straightforward way. The algorithm starts by solving the bottom right k x k block system. Note that since the diagonal blocks are diagonal matrices, this amounts to one single arithmetic division in each processor with each processor winding up with one component of x_m, the bottom k-dimensional block of the solution. In a general step j, we first need to deliver those just computed k components to each processor. This can be achieved by rotating the data x_{i+1} one element at a time around the ring, and requires a total time of

$$k(\beta_R + \tau_R).$$

Then we perform the vector operation:

$$b_j := b_j - B_j \hat{x}_j \tag{3.14}$$

where $\hat{x}_j$ is the vector consisting of the ν components following those of the j^{th} block x_j. while B_j is the rectangular matrix of size k x ν in block position $j, j+1$. The vector operation (3.14) consists in k independent vector operations each of length ν. Neglecting the cost of solving the diagonal systems, the arithmetic time for a typical step is therefore

$$\gamma + \nu\omega.$$

Summing the above times over the $m = N/k$ steps, we get that the total time for solving the system comes to

$$t_{T,R} = N(\beta_R + \tau_R) \tag{3.15}$$

for communication and

$$t_A = \frac{N}{k}[\nu\omega + \gamma], \tag{3.16}$$

for arithmetic. Observe that neither time increases as the number of processors k increases. Thus, it always pays to us as many processors as possible. Note, however, that for large ν the total time for solving the triangular system is small as compared with the time for Gaussian elimination.

4. The block interleaving algorithm for grid arrays

For convenience we define $\kappa \equiv \sqrt{k}$ and consider the $\kappa \times \kappa$ grid of processors of Figure 2. We number the processors with two indices as they would be numbered in a matrix: processor $P_{i,j}$ is the processor located at the intersection of the i^{th} row of the grid (starting from top) and the j^{th} column (starting from left). We partition A into square sub-matrices $A_{i,j}$ of size $\frac{\nu}{\kappa}$ x $\frac{\nu}{\kappa}$ and for simplicity we assume that A is a block-banded matrix of block-bandwidth κ i.e., we have $A_{i,j} = 0$ if $|i-j| > \kappa$. Note that for a general banded matrix of half-bandwidth ν this assumption is always satisfied by redefining ν to be $\nu_{new} \equiv \kappa\lceil \nu/(\kappa-1) \rceil$.

We proceed in two steps to assign the matrix to the processors:

1. We partition A into square sub–blocks of size ν x ν.
2. Each ν x ν sub–block is itself partitioned into square blocks of size $\frac{\nu}{\kappa}$ x $\frac{\nu}{\kappa}$ each. This partitioning is then mapped naturally onto the grid of processors, i.e., the (i,j) sub–block of each ν x ν block is assigned to the processor numbered (i,j). Clearly we need only store the nonzero sub–blocks.

In other words, relative to the partitioning of A into sub–matrices of size $\frac{\nu}{\kappa}$ x $\frac{\nu}{\kappa}$, the submatrix $A_{i,j}$ belongs to processor $P_{\Pi(i),\Pi(j)}$, where

$$\Pi(i) \equiv 1 + \mathrm{Mod}(i,\kappa) \quad \text{and} \quad \Pi(j) \equiv 1 + \mathrm{Mod}(j,\kappa). \tag{4.1}$$

Moreover, the element a_{ij} of A belongs to processor $P_{\pi(i),\pi(j)}$, where

$$\pi(i) \equiv \Pi(\lceil \frac{i}{\nu/\kappa} \rceil) \quad \text{and} \quad \pi(j) \equiv \Pi(\lceil \frac{j}{\nu/\kappa} \rceil). \tag{4.2}$$

This is illustrated for $\kappa = 4$ in Figure 6. With this scattering of the data each row of the matrix is distributed among the κ nodes of one row of the grid and each column is also distributed among

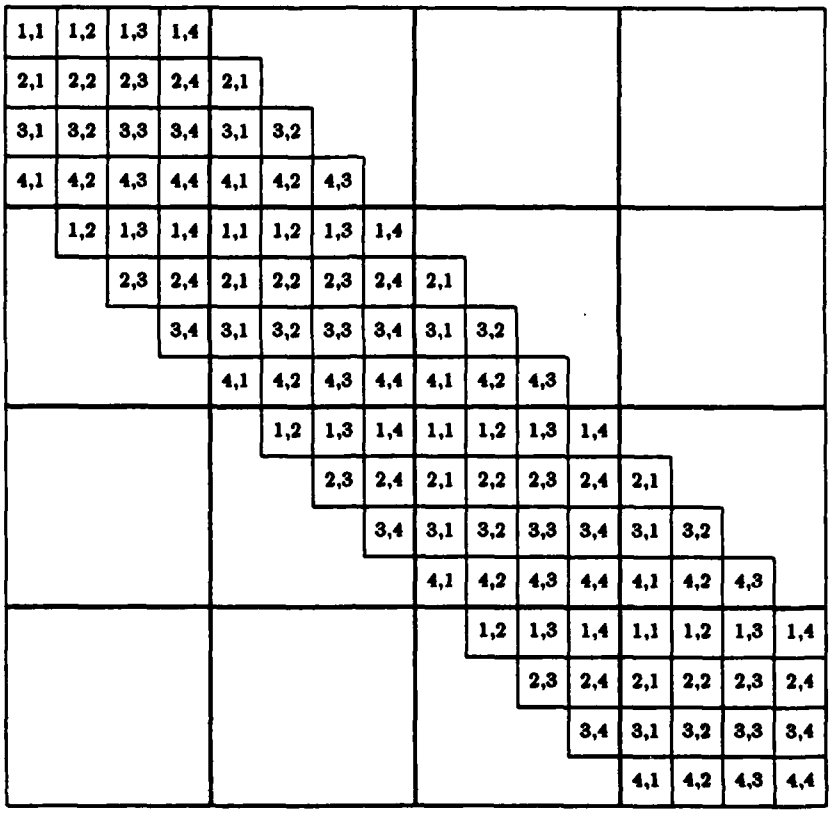


Figure 6: Block interleaving of a banded system in a 4 x 4 processor grid.

the nodes of one column of the grid. In order to avoid confusion between the rows and columns of the matrix and those of the multiprocessor grid, we will refer to the rows (resp., columns) of the grid as the grid-rows (resp., grid-columns). With this terminology, the Gaussian Elimination algorithm can naturally be implemented as follows:

ALGORITHM BIGGE (Banded Interleaved Grid Gaussian Elimination)

For $i = 1, 2, \ldots N-1$ do:

(1) ***Normalize row:*** Send $a_{i,i}$ to all processors of the same grid-row and divide the i^{th} row by a_{ii}, in each of these processors.

(2) ***Broadcast normalized pivot row vertically:*** All processors containing part of row i, i.e., processors $(\pi(i), *)$ send their part of the pivot row to all processors $(h, *), h = 1, 2, ..\kappa, h \neq \pi(i)$ of the same grid-column.

(3) ***Broadcast column of multipliers horizontally:*** Processors containing part of column i, i.e., processors $(*, \pi(i))$ send their part of the column of multipliers to all processors $(*, l), l = 1, 2, ..\kappa, l \neq \pi(i)$ of the same grid-row.

(4) ***Perform eliminations:*** Using the multipliers and the pivot row, each processor performs its part of the i^{th} step of Gaussian elimination.

The following result estimates the time of executing algorithm BIGGE.

Proposition 4.1. *If $\kappa \leq \nu$, the total time for performing Algorithm BIGGE is approximately*

$$t_{BIGGE} \approx N\left[\left(\frac{\nu}{\kappa}\right)^2 \omega + \frac{\nu}{\kappa}\gamma\right] + N\frac{\nu}{\kappa}\tau_G\left[1 + \kappa\alpha_G\right]^2 \tag{4.3}$$

$$= N\left[\left(\frac{\nu}{\kappa}\right)^2 \omega + \frac{\nu}{\kappa}\gamma\right] + N\left[\frac{\nu}{\kappa}\tau_G + \sqrt{2\nu\tau_G\beta_G} + \frac{1}{2}\beta_G\kappa\right],$$

where $\alpha_G \equiv \sqrt{\frac{\beta_G}{2\nu\tau_G}}$.

Proof. First, we discuss the time for data communication. We will neglect the lower order cost of communicating the element a_{ii} to processors $(\pi(i), *)$ in step (1). Steps (2) and (3) can be overlapped according to our assumptions (if not just double the communication time in the above result). Hence, steps (2) and (3) are essentially two overlapped broadcast operations, one in a vertical ring of κ processors and the other in a vertical ring of κ processors. By virtue of formula (2.8), both steps (2) and (3) can therefore be accomplished in a total time of

$$t_T(i) \approx \left(\sqrt{\frac{\nu}{\kappa}\tau_G} + \sqrt{\frac{\kappa}{2}\beta_G}\right)^2 \tag{4.4}$$

Summing the above times over the $N-1$ steps, we get the total time for data communication.

$$t_T \approx N\left(\sqrt{\frac{\nu}{\kappa}\tau_G} + \sqrt{\frac{\kappa}{2}\beta_G}\right)^2 \approx N\frac{\nu}{\kappa}\tau_G\left(1 + \sqrt{\frac{\kappa^2}{2\nu}\frac{\beta_G}{\tau_G}}\right)^2$$

$$\approx N\frac{\nu}{\kappa}\tau_G\left(1 + \frac{\kappa}{\nu}\sqrt{\nu\frac{\beta_G}{2\tau_G}}\right)^2.$$

Second, we discuss the time for arithmetic. Again we can neglect the cost of arithmetic operations of step (1) as they are of lower order. In the elimination step (4) each processor performs $\frac{\nu}{\kappa}$ consecutive operations of the form *row* := *row* − *scalar* * *row*, where *row* is a row of length $\frac{\nu}{\kappa}$. Hence, according to formula (2.4) the total arithmetic time for step i is

$$t_A(i) = \frac{\nu}{\kappa}(\frac{\nu}{\kappa}\omega + \gamma)$$

and for $N-1$ steps this comes to approximately:

$$t_A = N\left[\left(\frac{\nu}{\kappa}\right)^2\omega + \frac{\nu}{\kappa}\gamma\right].$$

Adding the communication and the arithmetic times we get the total of (4.3).

∎

A disappointing, but expected result, is that when the number of processors gets sufficiently large, *the total execution time increases due to the larger number of start-ups in communication* as is reflected by the last term of (4.3).

This raises the question of the optimal nunber of processors. Given an arbitrarily large number of processors (up to $k_{max} = \nu^2$) is it best to use all of the available processors or to "turn-off" a few of them and use only part of the available computing power? Although this seems counterintuitive, we show that in general, fewer than ν^2 processors must be used in order to get the maximum speed-up. For this purpose let us expand the total time (4.3) in terms of the variable $x \equiv \frac{\nu}{\kappa}$. We obtain

$$t_{BIGGE} \approx t(x) \equiv N\left[x^2\omega + (\gamma + \tau_G)x + \frac{1}{2}\frac{\nu\beta_G}{x}\right] + N\sqrt{2\nu\tau_G\beta_G}. \tag{4.5}$$

Note that the last term is constant with respect to the variable x. We would like to minimize the expression in (4.5) with respect to the real variable x. However, an attempt to do so would immediately lead to a third degree equation which would be difficult to solve. Fortunately, we can derive an approximate solution which will be shown to be nearly optimal. The near optimal solution is obtained by discarding the first degree term in x in (4.5), i.e., we seek the minimum of the following lower bound of $t(x)$:

$$\underline{t}(x) = N\left[x^2\omega + \frac{1}{2}\frac{\nu\beta_G}{x}\right] + N\sqrt{2\nu\tau_G\beta_G}. \tag{4.6}$$

Differentiating with respect to x, it is found that the minimum of $\underline{t}(x)$ is reached for

$$x_* = \left(\frac{\nu\beta_G}{4\omega}\right)^{1/3}. \tag{4.7}$$

Substituting in (4.6) we get

$$\underline{t}(x) \geq t_* \equiv \underline{t}(x_*) = 3N\omega^{1/3}\left(\frac{\beta_G\nu}{4}\right)^{2/3} + N\sqrt{2\nu\tau_G\beta_G} \quad \forall x, \quad t_* = 3N\omega x_*^2 + N\sqrt{2\nu\tau_G\beta_G}. \tag{4.8}$$

Note that the restriction $1 \leq \kappa \leq \nu^2$ translates into the restriction $\frac{1}{\nu} \leq x \leq \nu$.

The next proposition establishes that for large ν, the time t_* is nearly equal to the minimum of $t(x)$.

Proposition 4.2. *Let t_{opt} be the minimum of $t(x)$, $x \geq 0$, and let t_* be defined by (4.8). Then*

$$0 \leq \frac{t_{opt} - t_*}{t_*} \leq \frac{4}{3} \frac{\gamma + \tau_G}{(4\omega)^{2/3} \beta_G^{1/3} \nu^{1/3}} \tag{4.9}$$

Proof. By definition

$$t_{opt} \leq t(x_*) = \underline{t}(x_*) + N(\gamma + \tau_G) x_* \equiv t_* + N(\gamma + \tau_G) x_*. \tag{4.10}$$

If x_{opt} is the value of x for which $t(x)$ is minimum, we also have:

$$t_{opt} = t(x_{opt}) \geq \underline{t}(x_{opt}) \geq \underline{t}(x_*) = t_*. \tag{4.11}$$

From (4.10) and (4.11) we get

$$0 \leq \frac{t_{opt} - t_*}{t_*} \leq N(\gamma + \tau_G) \frac{x_*}{t_*} \tag{4.12}$$

The relation between t_* and x_* given by (4.8) yields

$$N \frac{x_*}{t_*} \leq \frac{1}{3\omega x_*} = \frac{4}{3} \frac{1}{(4\omega)^{2/3} \beta_G^{1/3} \nu^{1/3}}. \tag{4.13}$$

This, together with (4.12), gives (4.9).

∎

A similar argument was used in [7] to analyse the communication complexity of the Gaussian elimination algorithm.

The proposition asserts that the minimum time for performing Algorithm BIGGE, is of the form $O(N\nu^{2/3})$, i.e. the best speed-up is of the order of $\nu^{4/3}$, instead of $O(\nu^2)$ as might have been expected. The reason why the best possible speed-up cannot be achieved is due to the term that increases with k in t_{BIGGE}, i.e., to communication start-up times. If the successive steps of Gaussian elimination are overlapped then it is possible to reduce the effect of start-up. Thus, for a large number of processors, better algorithms can be found as will be seen in Section 6.

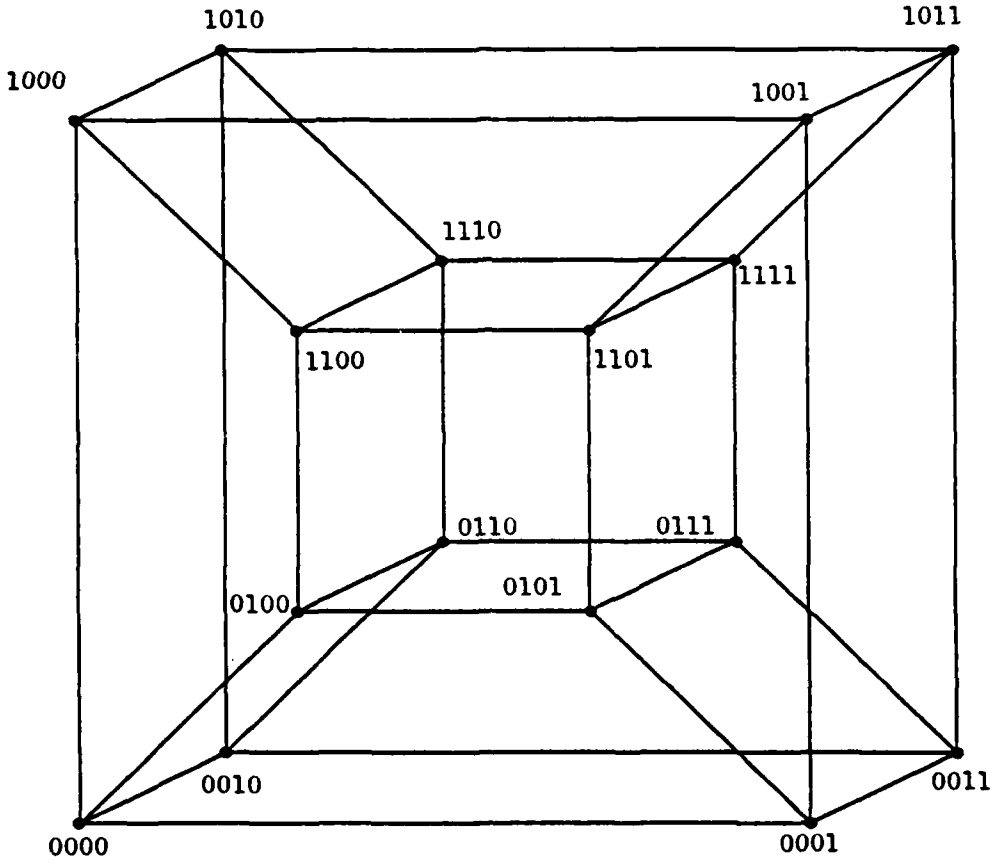


Figure 7: 3-D view of the 4-cube.

5. Gaussian elimination for the hypercube multiprocessor

The hypercube [13] is a popular ensemble architecture. The $n-$cube multiprocessor consists of 2^n nodes that are numbered by n-bit binary numbers, from 0 to $2^n - 1$. The interconnection is such that there is a direct link between two processors if and only if their binary representations differ by one and only one bit. In the case $n = 3$, the 8 nodes can be represented as the vertices of a three–dimensional cube, see Figure 3. When $n = 4$ the three–dimensional representation shown in Figure 7 is quite useful. In Figure 7, a 4-cube is obtained by joining all nodes of an inner–3–cube with their geometrical counterparts of an outer–3–cube. Some general results on the topology of hypercubes are discussed in [8] while reference [9] examines communication problems in a hypercube multiprocessor.

5.1. The Hypercube Banded Gaussian Elimination

For this section we assume that n is even. The number of processors, denoted by k, is of the form $k = 2^n$ and, as in Section 4, we define $\kappa = \sqrt{k} = 2^{n/2}$. In order to be able to derive an analogue of Algorithm BIGGE for the hypercube, we first map a κ x κ grid into the n–cube and then map the algorithm onto the defined grid. However, we will try to exploit the more advantageous interconncetion features of the hypercube, when moving data in algorithm BIGGE.

To map 2-D a grid into n-cubes, we make the important observation that an n–cube can be viewed as a "cross-product" of two $\frac{n}{2}$–cubes. Indeed, we can consider the n-bit binary label of any n–cube node as the result of concatenating two $\frac{n}{2}$–bit binary numbers, say b_i and c_j. In other words, we can write any node number as $a_{ij} = b_i \wedge c_j$, where $\wedge$ denotes the concatenation, and b_i, c_j are the first and second $\frac{n}{2}$ bits of the node number. From the properties of the n–cube it can be easily seen that when b_j is fixed the resulting $2^{\frac{n}{2}}$ nodes obtained by varying the second part of the binary number, i.e., by varying c_i, form an $\frac{n}{2}$–cube, i.e., a sub-cube of the n-cube. Similarly, when we fix the second part c_i and let b_j vary, we obtain another $\frac{n}{2}$–subcube.

This defines in a natural way the n–cube as a cross–product of the two $\frac{n}{2}$-cubes. We refer to a vertical plane as an $\frac{n}{2}$–cube defined as the set of all a_{ij} where j is fixed. A horizontal plane is defined likewise by fixing i and letting j vary. This is illustrated in Figure 7 where the 4 horizontal planes are numbered with one of the four binary numbers 00, 01, 11, 10 (in this order) from bottom to top. Then the vertical planes defined by the 4 vertical trapezes are in turn successively numbered 00, 01, 11, 10. Each node of the 4-cube is the only intersection point of a horizontal plane and a vertical plane. Note that this numbering is by no means unique.

Let us now assume, as was done in Section 4, that the matrix A is a block-banded matrix of block-bandwidth κ, where each block is of size ν/κ, i.e., we have $A_{i,j} = 0$ if $|i - j| \geq \kappa$. If we partition A into sub-matrices of size $\frac{\nu}{\kappa}$ x $\frac{\nu}{\kappa}$, then the submatrix $A_{i,j}$ is assigned to the node numbered $h(i) \wedge h(j)$, where $h(i) \equiv Binary[\text{Mod}(i, \kappa)]$. The distribution of Figure 6 is still valid but the processors on the same block-column, now form an $\frac{n}{2}$–cube while those of the same block–row form another $\frac{n}{2}$-cube.

The only difference with the grid algorithm is that communication is faster because of the richer interconnections of the subcubes. In what follows we will make the assumption that a node can be crossed simultaneously by two streams of data, one going vertically, i.e., traveling in the vertical plane of the node, while the other travels in the horizontal plane of the node. It is not necesseary to rewrite the algorithm as it is identical with Algorithm BIGGE. We refer to the new algorithm as the Hypercube Banded Gaussian Elimination (HBGE).

We now estimate the time required to perform Algorithm HBGE. The time for performing arithmetic operations is identical with that of BIGGE, i.e., it is given by

$$t_{a,HBGE} = N\left[\left(\frac{\nu}{\kappa}\right)^2 \omega + \gamma\frac{\nu}{\kappa}\right]. \tag{5.1}$$

In order to estimate the time required for the communication tasks, all we need to know is the time for moving a vector of length m from one processor to all others in an $\frac{n}{2}$–cube, as this is the only important transfer operation of the algorithm, which takes place in steps (2) and (3) (simultaneously). This was examined in detail in Section 2.2, and as a result we infer the following Proposition.

Proposition 5.1. *The total time required to perform Algorithm HBGE on a hypercube of $k \equiv \kappa^2$ processors where κ is a power of 2, is approximately*

$$t_{HBGE} \approx N\left[\left(\frac{\nu}{\kappa}\right)^2\omega + \gamma\frac{\nu}{\kappa}\right] + N\frac{\nu}{\kappa}\tau_H\left(1+\alpha_H\sqrt{2\kappa log_2\kappa}\right)^2 \tag{5.2}$$

$$= N\left[\left(\frac{\nu}{\kappa}\right)^2\omega + \frac{\nu}{\kappa}\gamma\right] + N\left[\frac{\nu}{\kappa}\tau_H + 2\sqrt{\nu\tau_H\beta_H\frac{log_2\kappa}{\kappa}} + \beta_H log_2\kappa\right].$$

where $\alpha_H = \sqrt{\frac{\beta_H}{2\nu\tau_H}}$.

Note that α_H has been chosen of this form in order to be consistent with previous results.

6. Pipelining and the wavefront approach for the grid architecture

As was observed in the previous sections, when the number of processors is large compared to the half-bandwidth ν, the loss of efficiency appears to be nonnegligible due to communication overhead. Thus, while one expected a speed-up of the order of ν^2 using the maximum allowable number of processors, $k = \nu^2$, we were only able to achieve a speed-up of the order of $\nu^{4/3}$. The reason for this inefficiency is that those algorithms do not pipeline the successive steps of Gaussian elimination. When a column or a row of information is broadcast, some processors are idle waiting for the information to reach them and since the number of processors is large the information will travel a long way across processors leading to important idle times. It is shown in this section that it is vital in this situation to resort to pipelining, i.e., to employ an algorithm of finer granularity and overlap the successive steps of the elimination as much as possible. There are a variety of ways of achieving this for the Gaussian elimination algorithm.

A systematic way of pipelining in Gaussian elimination is the so-called wavefront approach, which is well–known in the context of algorithms for systolic architectures [6]. We describe here an adaptation of this approach for solving banded linear systems. The algorithm can be viewed as a simple implementation of the banded *LU* factorization described in pages 280-285 of [6], which seems to have been known for a long time [10].

Put in very simple terms, the wavefront approach consists in performing the elimination by skew-diagonals, a skew-diagonal of A being the set of elements $a_{i,j}$ such that $i+j$ is equal to some constant. In other words, all computations relative to an elimination step of Gauss' algorithm are performed simultaneously on the elements of the matrix that are on the same skew-diagonal. The basic idea of the wavefront approach is that we can overlap several consecutive such computations which are referred to as waves. To describe the algorithm in its simplest form, we initially assume that $k = \nu^2$, i.e., each box of Figure 6, represents a processor which contains one nonzero element of A. For the following discussion, we refer to both Figure 6 for the matrix assignment and Figure 8 for an illustration of the successive steps. A general step will consist of a communication task and a computation task. In the very first step of the algorithm, processor $(1,1)$ sends the element $a_{1,1}$ downward to processor $(2,1)$, which then uses this information to compute the multiplier $l_{2,1} \equiv a_{2,1}/a_{1,1}$. In the second step, this multiplier is now transferred to the processor $(2,2)$ to the right. Simultaneously, processor $(2,1)$ passes on the element $a_{1,1}$ that it has received in step 1, downward to processor $(3,1)$, while processor $(1,2)$ transmits the element $a_{1,2}$ downward to processor $(2,2)$. After these data transfers, processors $(1,3),(2,2)$, and $(3,1)$ perform in parallel the elimination work corresponding to the first row of the matrix, on the elements $a_{1,3}, a_{2,2}$ and $a_{3,1}$. Thus, processor $(1,3)$ is actually idle (no work is done on the first row which is the pivot row), processor $(2,2)$ computes the new element $a^1_{2,2}$ resulting from the elimination of row 1, using

the data $a_{2,2}$, $a_{1,2}$, and $l_{2,1}$ and processor $(3,1)$ computes the multiplier $l_{3,1} \equiv a_{3,1}/a_{1,1}$. One can distinguish the formation of a first wave of computations.

In the next step (step 3) this wave propagates to the right by transferring the multipliers $l_{i,1}, i \equiv 2,3$ to the eastern neighbors and the first row elements downward to the southern neighbors. The elimination corresponding to the first row is then performed on the fourth skew diagonal and the new multiplier $l_{4,1} = a_{4,1}/a_{1,1}$ is formed in processor $(4,1)$. A general step consists of transferring the multipliers l_{ik} to the right and the pivot row elements downward, and then performing arithmetic, i.e., computing the reduced elements. At the fourth step, the second wave corresponding to the elimination by the second row can be started. Indeed, the last task of the elimination of row 1 has just been completed on the elements $a_{2,2}$ and $a_{2,3}$ and therefore the multiplier $l_{3,2}$ can be computed in processor $(3,2)$. Here, we need to send not only the elements of the first wave to the right but also the element $a_{1,2}$ downward from processor $(2,2)$ to processor $(3,2)$.

It is easy to deduce the following steps. The first 8 steps of the algorithm are illustrated in Figure 8 for the case $\nu = 4$. A new wave appears every three steps and dies off after a total of $2(\nu - 1)$ steps. If we refer to a front as the set of the concurrently active waves, the wavefront consists of a total of $\nu - 2$ waves, dealing with $\nu - 2$ consecutive pivot rows, at the high regime of the pipelining, i.e., after the first $2\nu - 2$ steps and before the last $\nu - 1$ steps.

New waves appear in steps $1, 4, 7, \ldots 3(i-1)+1, \ldots$. The wave carrying the information for the elimination by row i, i.e. for forming the matrix $A^{(i)}$, appears in step $3(i-1)+1$. In particular, for $i = N$, we would be starting a fictious wave corresponding to the elimination of row N, while finishing the work on element $a_{N,N}^{N-1}$ by the previous wave. In other words the algorithm stops in step $3(N-1)+1 = 3N-2$. A comparison with previous work on (dense) wavefront factorization reveals that the method requires exactly the same number of steps [10]. Therefore, the only difference with the dense case is that we are using a smaller number of processors, because of the bandedness of the matrix.

With the assignment of Figure 6, the active processors are performing the same amount of work at any given step. Typically, in the same wave, some processors will be performing an elimination of the form

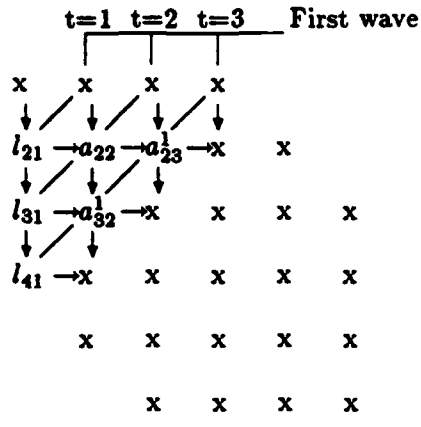
$$a_{ij}^{k+1} = a_{ij}^{k} - l_{ik} a_{kj}^{k} \tag{6.1}$$

while (at most one) will compute a new multiplier l_{ik}. Looking at communication patterns each step requires simultaneous nearest neighbor communication whereby any processor of the same wave sends multipliers eastward and pivot-row elements southward. These data transfers can be overlapped. As a result, each step costs a total of $\gamma+\omega$ for arithmetic and $\beta_G+\tau_G$ for communication. Therefore, the wavefront algorithm for factoring a banded matrix requires a total of

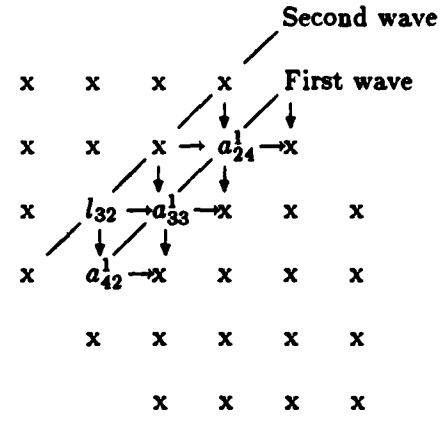
$$(3N-2)\left[\gamma + \omega + \beta_G + \tau_G\right].$$

A remarkable fact is that this number does not depend on the bandwidth of the matrix. However, any complexity result based on this observation is deceiving since the number of processors required to achieve this time increases like the square of the half-bandwidth. It is unlikely that the problems encountered in nature will exactly fit some priviledged number k of processors of a given configuration. A less unrealistic goal is to solve *any problem* by choosing a solution method according to various parameters of that problem, such as its size and its bandwidth in the present case. For example, if the number of processors is much larger than ν^2, one might use a cyclic reduction type algorithm [4], or a substructured Gaussian elimination method [1] which are not considered here.

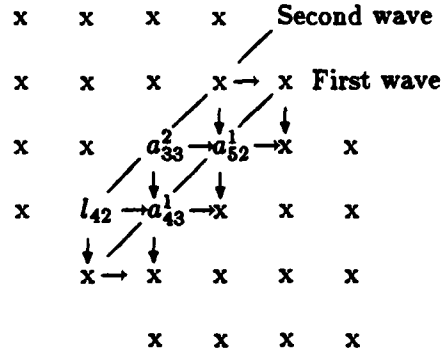
As a consequence we now assume that the number of processors is less than ν^2. The above algorithm can be generalized by simply partitioning the matrix in blocks of size $\frac{\nu}{\kappa}$ each. The block



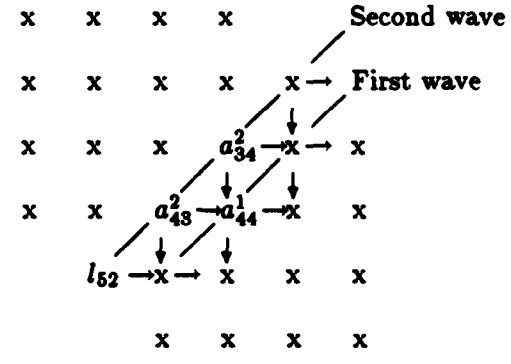
Steps 1 to 3



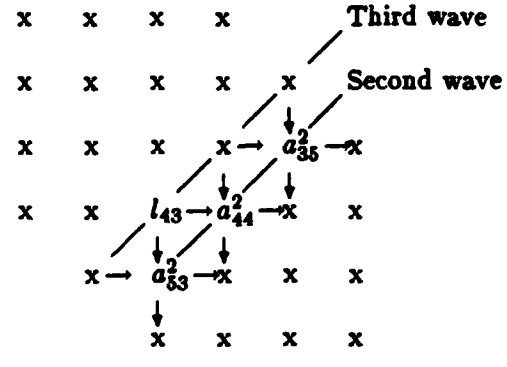
Step 4



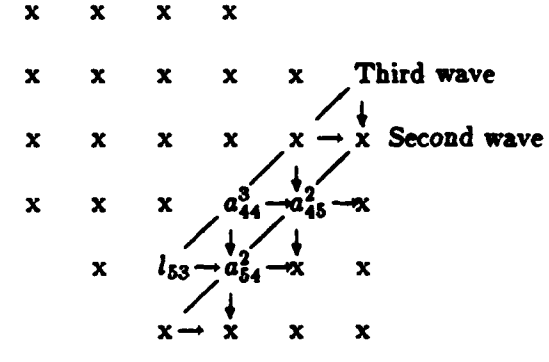
Step 5



Step 6



Step 7



Step 8

Figure 8: Wavefront algorithm for a banded linear system.

matrix resulting from this partitioning is of block-size $N/(\nu/\kappa) = \frac{N\kappa}{\nu}$. The adaptation of the wavefront method is straightforward. The multipliers, now matrices of size $\frac{\nu}{\kappa}$, require multiplying the inverse of a matrix by another matrix, in each wavefront step. Likewise, the typical operation (6.1) becomes a matrix operation. It can be seen that each step is dominated by the cost of the matrix multiplication, i.e., by the time $(\frac{\nu}{\kappa})^2(\gamma + \frac{\nu}{\kappa}\omega)$. Each step of communication on the other hand requires a time of $\beta_G + \left(\frac{\nu}{\kappa}\right)^2 \tau_G$. Summing up over the $3N\kappa/\nu - 2$ steps of the algorithm, we find an approximate total time of

$$t_{W,Block} \approx 3N\frac{\kappa}{\nu}\left[\left(\frac{\nu}{\kappa}\right)^2(\gamma + \frac{\nu}{\kappa}\omega) + \beta_G + \left(\frac{\nu}{\kappa}\right)^2\tau_G\right]$$

or,

$$t_{W,Block} \approx 3N\left[\frac{\nu}{\kappa}\left(\gamma + \frac{\nu}{\kappa}\omega\right) + \frac{\kappa}{\nu}\beta_G + \frac{\nu}{\kappa}\tau_G\right].$$

A quick comparison with the ring and the grid algorithms of the previous sections indicates that for small values of κ, the wavefront algorithm will be about three times slower. However, the wavefront algorithm allows us to increase the number of processors to ν^2 and still attain linear speed-up.

It is important to say a few words about the solution of the resulting triangular system. A wavefront-like method has been devised for solving (dense) triangular systems in [3]. The method called TCB/G (Triangular system solution by Column - Blocks/Greedy) consists of sweeping from bottom to top by marching along skew diagonals. The method can be easily adapted to banded linear systems with essentially no modification to it, except for the fact that the processors are now assigned in a manner that accounts for bandedness. In contrast with the LU factorization there is only one wave moving and the total number of scalar steps is $2N - 1$. Therefore, the time for solving the triangular system by this wavefront method is comparable with the time to perform the Gaussian reduction to triangular form. The obvious reason is that processors are idle during most of the algorithm.

7. Summary of results and conclusion

The complexity results of four Gaussian elimination algorithms are summarized in Table 1. The optimal time for the ring shown in the table is in reality the upper bound given by (3.13). For Algorithm HBGE the optimal time is difficult to compute. The time shown in the table is an upper bound obtained by letting $\kappa = \nu$, i.e., by using the maximun possible number of processors. This will be nearly optimal in general as the total time is the sum of a decreasing function of κ and a logarithmicly increasing function of k.

To conclude, we can make the following observations:

- The time for performing arithmetic is the same for the first three methods and three times as long for the fourth method for ν large enough.
- The time for communication shows a decrease by a factor of $\sqrt{k}$ when passing from a linear ring to a grid. This interesting property is due to the fact that the bandwidth in the vertical and horizontal directions is multiplied by $\sqrt{k}$. Also latency times are reduced because the diameter of the grid is of the order of $\sqrt{k}$ times smaller than that of the ring.
- For the hypercube topology the latency term is of the form $N\beta_G log_2 k$. The consequence of this is that the deterioration of efficiency due to communication overhead grows much more slowly than that of the ring or the grid.
- For a large number of processors, in all methods which do not use pipelining (i.e., the first three methods) the degradation in performance due to communication overhead is important and the wavefront approach is to be preferred, as is reflected by the last column of the table.

Method	t_{Arithm}	t_{Comm}	t_{opt}
RIBGE	$N\lceil\frac{\nu-1}{k}\rceil(\nu\omega+\gamma)$	$N\nu\tau_R\left[1+\kappa\alpha\right]^2$	$2N\nu\left[\sqrt{(\omega+\gamma/\nu)\beta_R}+\tau_R\right]$
BIGGE	$N\left[\left(\frac{\nu}{\kappa}\right)^2\omega+\gamma\frac{\nu}{\kappa}\right]$	$N\frac{\nu}{\kappa}\tau_G\left[1+\kappa\alpha\right]^2$	$3N\omega^{1/3}\left[\frac{\beta_G\nu}{4}\right]^{2/3}+N\sqrt{2\nu\beta_G\tau_G}$
HBGE	$N\left[\left(\frac{\nu}{\kappa}\right)^2\omega+\gamma\frac{\nu}{\kappa}\right]$	$N\frac{\nu}{\kappa}\tau_H\left[1+\sqrt{2\kappa log_2\kappa}\alpha\right]^2$	$N\left[(\omega+\gamma)+\tau_H+\beta_H log_2(\nu)\right]$
WFGE	$3N\frac{\nu}{\kappa}\left[\gamma+\frac{\nu}{\kappa}\omega\right]$	$3N\left[\frac{\kappa}{\nu}\beta+\frac{\nu}{\kappa}\tau\right]$	$3N\left[\gamma+\omega+\beta+\tau\right]$

Note : $\alpha = \sqrt{\frac{\beta}{2\nu\tau}}$. where τ, β are either τ_R, β_R (ring) or τ_G, β_G (grid), or τ_H, β_H (cube).

Table 1: Timings for parallel Gaussian elimination algorithms

- For small number of processors the wavefront method does not perform as well as the simpler nonpipelined methods.
- The optimal times in which a banded linear system can be solved by the nonpipelined Gaussian elimination, if we had an arbitrarily large number of processors, is of the form $O(N\nu)$ for the ring, $O(N\nu^{2/3})$ for the 2–grid and $O(Nlog_2(\nu))$ for the cube. Comparing with the $O(N\nu^2)$ time of a single processor, we observe that the important change takes place when passing from the 2–D grid architecture to the hypercube architecture.

References

[1] J.J. Dongarra and A. Sameh, *On some parallel banded system solvers,* Technical Report MCSD, No. 27, Argonne National Lab., 1984.

[2] D. Gannon, J. van Rosendale, *On the Impact of Communication Complexity in the Design of Parallel Algorithms,* Technical Report 84-41, ICASE, 1984.

[3] I. Ipsen, Y. Saad, M.H. Schultz, *Complexity of dense linear system solution on a multiprocessor ring,* Technical Report 349, Computer Science Dept., Yale University, 1984.

[4] L. S. Johnsson, *Odd-Even cyclic reduction on ensemble architectures.,* Technical Report YALEU/DCS/RR-339, Computer Science Dept., Yale University, 1984.

[5] ————, *Fast Banded Systems solvers for Ensemble Architectures.,* Technical Report YALEU/DCS/RR-379, Computer Science Dept., Yale University, 1985.

[6] C.A. Mead and L.A. Conway, *Introduction to VLSI Systems,* Addison Wesley, Mass.,USA, 1980.

[7] Y. Saad, *Communication complexity of the Gaussian elimination algorithm on multiprocessors.,* Technical Report YALEU/DCS/RR-348, Yale University, 1983.

[8] Y. Saad, M.H. Schultz, *Topological properties of hypercubes,* Technical Report YALEU/DCS/RR-389, Computer Science Dept., Yale University, 1985.

[9] ————, *Data Communication in Hypercubes,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[10] A.H. Sameh, Numerical Parallel Algorithms - A Survey, D. Lawrie, A. Sameh ed., *High Speed Computer and Algorithm Organization,* Academic Press, New York, 1977, pp. 207–228.

[11] ————, *On Some Parallel Algorithms on a Ring of Processors,* Technical Report , University of Illinois at Urbana-Champaign, 1984.

[12] M.H. Schultz, *Multiple Array Processors for Ocean Acoustic Problems,* Technical Report YALEU/DCS/ RR-363, Dept. of Computer Science, Yale University, 1985.

[13] C.L. Seitz, *The Cosmic Cube,* CACM, 28 (1985), pp. 22–33.

END

FILMED

11-85

DTIC